Indira Gandhi Delhi Technical University for Women

**(Established by Govt. of Delhi vide Act 09 of 2012)**

**(Formerly Indira Gandhi Institute of Technology)**

**Kashmere Gate, Delhi - 110006**



**LABORATORY FILE FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS**

**MCA-201**

**SUBMITTED TO: SUBMITTED BY: MS. Bhawna Narwal Srishti Jain**

**(02204092019)**

**8. WAP to implement Quicksort**

#include <iostream>

using namespace std;

// Function to swap position of elements

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

// Function to print elements of an array

void printArray(int array[], int size) {

int i;

for (i = 0; i < size; i++)

cout << array[i] << " ";

cout << endl;

}

// Function to partition the array on the basis of pivot element

int partition(int array[], int low, int high) {

// Select the pivot element

int pivot = array[high];

int i = (low - 1);

// Put the elements smaller than pivot on the left

// and greater than pivot on the right of pivot

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int array[], int low, int high) {

if (low < high) {

// Select pivot position and put all the elements smaller

// than pivot on left and greater than pivot on right

int pi = partition (array, low, high);

// Sort the elements on the left of pivot

quickSort(array, low, pi - 1);

// Sort the elements on the right of pivot

quickSort(array, pi + 1, high);

}}

// Driver code

int main () {

int n ;

cout<<"Enter the size of the array : ";

cin>>n;

int a[n];

cout<<"Enter the elements of the array : ";

for(int i=0;i<n;i++)

{

cin>>a[i];

}

quickSort(a, 0, n - 1);

cout << "Sorted array in ascending order: \n";

printArray(a, n);

}

**Worst Case Complexity [Big-O]: O(n^2)**

It occurs when the pivot element picked is either the

greatest or the smallest element.

This condition leads to the case in which the pivot

element lies in an extreme end of the sorted array. One

sub-array is always empty and another sub-array contains

n - 1 elements. Thus, quicksort is called only on this sub

array.

However, the quick sort algorithm has better

performance for scattered pivots.

**Best Case Complexity [Big-omega]: O (n\*log n)**

It occurs when the pivot element is always the middle

element or near to the middle element.

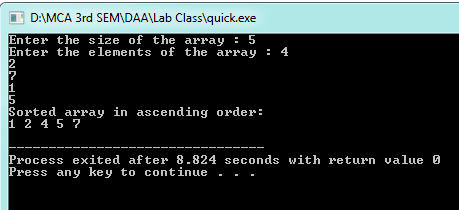
**Average Case Complexity [Big-theta]: O (n\*log n)**It occurs when the above conditions do not occur.

**Space Complexity :O (log n).**

**The recurrence relation is: T(n) = T(k) + T(n-k-1) +**

**theta(n)**

**Output:**



**9. WAP to implement merge sort**

**Solution:**

#include <iostream>

using namespace std;

// Merge two subarrays L and M into arr

void merge(int arr[], int p, int q, int r) {

// Create L ? A[p..q] and M ? A[q+1..r]

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

// Maintain current index of sub-arrays and main array

int i, j, k;

i = 0;

j = 0;

k = p;

// Until we reach either end of either L or M, pick larger among

// elements L and M and place them in the correct position at A[p..r]

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

// When we run out of elements in either L or M,

// pick up the remaining elements and put in A[p..r]

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[j];

j++;

k++;

}}

// Divide the array into two subarrays, sort them and merge them

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// m is the point where the array is divided into two subarrays

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted subarrays

merge(arr, l, m, r);

}}

// Print the array

void printArray(int arr[], int size)

{

for (int i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

// Driver program

int main()

{

int n;

cout<<"Enter the size of the array : ";

cin>>n;

int a[n];

cout<<"Enter the elements of the array : ";

for(int i=0;i<n;i++)

{

cin>>a[i];

}

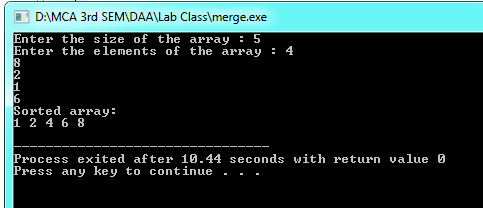
mergeSort(a, 0, n - 1);

cout << "Sorted array: \n";

printArray(a, n);

return 0;

}



**Merge Sort Time Complexity**

**Best Case Complexity:** O(n\*log n)

**Worst Case Complexity:** O(n\*log n)

**Average Case Complexity:** O(n\*log n)

**Space Complexity**The space complexity of merge sort is O(n).

T(n) = 2T(n/2) + Theta(n)

**Sorting in Place:** No in a typical implementation

1. **Implement Insertion Sort**

// Recursive C++ program for insertion sort

#include <iostream>

using namespace std;

// Recursive function to sort an array using

// insertion sort

void insertionSortRecursive(int arr[], int n)

{

// Base case

if (n <= 1)

return;

// Sort first n-1 elements

insertionSortRecursive( arr, n-1 );

// Insert last element at its correct position

// in sorted array.

int last = arr[n-1];

int j = n-2;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > last)

{

arr[j+1] = arr[j];

j--;

}

arr[j+1] = last;

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

for (int i=0; i < n; i++)

cout << arr[i] <<" ";

}

/\* Driver program to test insertion sort \*/

int main()

{

int n;

cout<<"Enter the size of the array : ";

cin>>n;

int a[n];

cout<<"Enter the elements of the array : ";

for(int i=0;i<n;i++)

{

cin>>a[i];

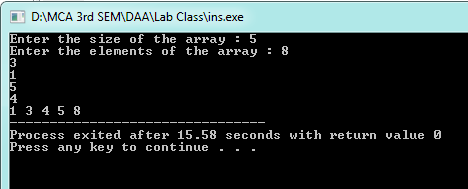
}

insertionSortRecursive(a, n);

printArray(a, n);

return 0;

}



The recurrence relation of the code of recursive insertion sort is T(n) = T(n-1) + n

T(n)=T(n-1)+n

=T(n-2)+(n-1)+n

=...

=θ(n2)

**12**.**Implement Bubble Sort**

// C/C++ program for recursive implementation

// of Bubble sort

#include <bits/stdc++.h>

using namespace std;

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

// Base case

if (n == 1)

return;

// One pass of bubble sort. After

// this pass, the largest element

// is moved (or bubbled) to end.

for (int i=0; i<n-1; i++)

if (arr[i] > arr[i+1])

swap(arr[i], arr[i+1]);

// Largest element is fixed,

// recur for remaining array

bubbleSort(arr, n-1);

}

/\* Function to print an array \*/

void printArray(int arr[], int n)

{

for (int i=0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int n;

cout<<"Enter the size of the array : ";

cin>>n;

int a[n];

cout<<"Enter the elements of the array : ";

for(int i=0;i<n;i++)

{

cin>>a[i];

}

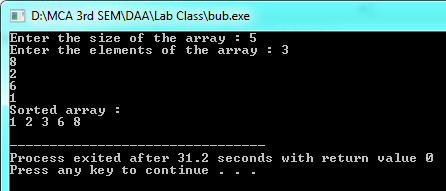
bubbleSort(a, n);

printf("Sorted array : \n");

printArray(a, n);

return 0;

}



The recurrence relation of the code of recursive bubble sort is T(n) = T(n-1) + n

T(n)=T(n-1)+n

=T(n-2)+(n-1)+n

=...

=θ(n2)